Systemy operacyjne 1

Zakleszczenia

Arkadiusz Chrobot

Katedra Systemów Informatycznych, Politechnika Świętokrzyska w Kielcach

Kielce, 14 listopada 2024

Plan wykładu

1. Zakleszczenia
   1. Definicja
   2. Dostęp do zasobów współdzielonych
   3. Warunki konieczne do wystąpienia zakleszczenia
   4. Metody opisu zakleszczeń
2. Zapobieganie zakleszczeniom
   1. Wzajemne wykluczanie
   2. Przetrzymywanie i oczekiwanie
   3. Brak wywłaszczeń
   4. Czekanie cykliczne
3. Unikanie zakleszczeń
   1. Algorytm bankiera
   2. Algorytm bezpieczeństwa
   3. Metoda dla zasobów reprezentowanych pojedynczo
4. Wykrywanie i wychodzenie z zakleszczeń
   1. Algorytm wykrywania zakleszczenia
   2. Metody wychodzenia z zakleszczenia
5. Łączenie metod postępowania z zakleszczeniami
6. Algorytm strusia

Plan wykładu

1. Zakleszczenia
   1. Definicja
   2. Dostęp do zasobów współdzielonych
   3. Warunki konieczne do wystąpienia zakleszczenia
   4. Metody opisu zakleszczeń
2. Zapobieganie zakleszczeniom
   1. Wzajemne wykluczanie
   2. Przetrzymywanie i oczekiwanie
   3. Brak wywłaszczeń
   4. Czekanie cykliczne
3. Unikanie zakleszczeń
   1. Algorytm bankiera
   2. Algorytm bezpieczeństwa
   3. Metoda dla zasobów reprezentowanych pojedynczo
4. Wykrywanie i wychodzenie z zakleszczeń
   1. Algorytm wykrywania zakleszczenia
   2. Metody wychodzenia z zakleszczenia
5. Łączenie metod postępowania z zakleszczeniami
6. Algorytm strusia

Plan wykładu

1. Zakleszczenia
   1. Definicja
   2. Dostęp do zasobów współdzielonych
   3. Warunki konieczne do wystąpienia zakleszczenia
   4. Metody opisu zakleszczeń
2. Zapobieganie zakleszczeniom
   1. Wzajemne wykluczanie
   2. Przetrzymywanie i oczekiwanie
   3. Brak wywłaszczeń
   4. Czekanie cykliczne
3. Unikanie zakleszczeń
   1. Algorytm bankiera
   2. Algorytm bezpieczeństwa
   3. Metoda dla zasobów reprezentowanych pojedynczo
4. Wykrywanie i wychodzenie z zakleszczeń
   1. Algorytm wykrywania zakleszczenia
   2. Metody wychodzenia z zakleszczenia
5. Łączenie metod postępowania z zakleszczeniami
6. Algorytm strusia

Plan wykładu

1. Zakleszczenia
   1. Definicja
   2. Dostęp do zasobów współdzielonych
   3. Warunki konieczne do wystąpienia zakleszczenia
   4. Metody opisu zakleszczeń
2. Zapobieganie zakleszczeniom
   1. Wzajemne wykluczanie
   2. Przetrzymywanie i oczekiwanie
   3. Brak wywłaszczeń
   4. Czekanie cykliczne
3. Unikanie zakleszczeń
   1. Algorytm bankiera
   2. Algorytm bezpieczeństwa
   3. Metoda dla zasobów reprezentowanych pojedynczo
4. Wykrywanie i wychodzenie z zakleszczeń
   1. Algorytm wykrywania zakleszczenia
   2. Metody wychodzenia z zakleszczenia
5. Łączenie metod postępowania z zakleszczeniami
6. Algorytm strusia

Plan wykładu

1. Zakleszczenia
   1. Definicja
   2. Dostęp do zasobów współdzielonych
   3. Warunki konieczne do wystąpienia zakleszczenia
   4. Metody opisu zakleszczeń
2. Zapobieganie zakleszczeniom
   1. Wzajemne wykluczanie
   2. Przetrzymywanie i oczekiwanie
   3. Brak wywłaszczeń
   4. Czekanie cykliczne
3. Unikanie zakleszczeń
   1. Algorytm bankiera
   2. Algorytm bezpieczeństwa
   3. Metoda dla zasobów reprezentowanych pojedynczo
4. Wykrywanie i wychodzenie z zakleszczeń
   1. Algorytm wykrywania zakleszczenia
   2. Metody wychodzenia z zakleszczenia
5. Łączenie metod postępowania z zakleszczeniami
6. Algorytm strusia

Plan wykładu

1. Zakleszczenia
   1. Definicja
   2. Dostęp do zasobów współdzielonych
   3. Warunki konieczne do wystąpienia zakleszczenia
   4. Metody opisu zakleszczeń
2. Zapobieganie zakleszczeniom
   1. Wzajemne wykluczanie
   2. Przetrzymywanie i oczekiwanie
   3. Brak wywłaszczeń
   4. Czekanie cykliczne
3. Unikanie zakleszczeń
   1. Algorytm bankiera
   2. Algorytm bezpieczeństwa
   3. Metoda dla zasobów reprezentowanych pojedynczo
4. Wykrywanie i wychodzenie z zakleszczeń
   1. Algorytm wykrywania zakleszczenia
   2. Metody wychodzenia z zakleszczenia
5. Łączenie metod postępowania z zakleszczeniami
6. Algorytm strusia

Plan wykładu

1. Zakleszczenia
   1. Definicja
   2. Dostęp do zasobów współdzielonych
   3. Warunki konieczne do wystąpienia zakleszczenia
   4. Metody opisu zakleszczeń
2. Zapobieganie zakleszczeniom
   1. Wzajemne wykluczanie
   2. Przetrzymywanie i oczekiwanie
   3. Brak wywłaszczeń
   4. Czekanie cykliczne
3. Unikanie zakleszczeń
   1. Algorytm bankiera
   2. Algorytm bezpieczeństwa
   3. Metoda dla zasobów reprezentowanych pojedynczo
4. Wykrywanie i wychodzenie z zakleszczeń
   1. Algorytm wykrywania zakleszczenia
   2. Metody wychodzenia z zakleszczenia
5. Łączenie metod postępowania z zakleszczeniami
6. Algorytm strusia

Plan wykładu

1. Zakleszczenia
   1. Definicja
   2. Dostęp do zasobów współdzielonych
   3. Warunki konieczne do wystąpienia zakleszczenia
   4. Metody opisu zakleszczeń
2. Zapobieganie zakleszczeniom
   1. Wzajemne wykluczanie
   2. Przetrzymywanie i oczekiwanie
   3. Brak wywłaszczeń
   4. Czekanie cykliczne
3. Unikanie zakleszczeń
   1. Algorytm bankiera
   2. Algorytm bezpieczeństwa
   3. Metoda dla zasobów reprezentowanych pojedynczo
4. Wykrywanie i wychodzenie z zakleszczeń
   1. Algorytm wykrywania zakleszczenia
   2. Metody wychodzenia z zakleszczenia
5. Łączenie metod postępowania z zakleszczeniami
6. Algorytm strusia

Plan wykładu

1. Zakleszczenia
   1. Definicja
   2. Dostęp do zasobów współdzielonych
   3. Warunki konieczne do wystąpienia zakleszczenia
   4. Metody opisu zakleszczeń
2. Zapobieganie zakleszczeniom
   1. Wzajemne wykluczanie
   2. Przetrzymywanie i oczekiwanie
   3. Brak wywłaszczeń
   4. Czekanie cykliczne
3. Unikanie zakleszczeń
   1. Algorytm bankiera
   2. Algorytm bezpieczeństwa
   3. Metoda dla zasobów reprezentowanych pojedynczo
4. Wykrywanie i wychodzenie z zakleszczeń
   1. Algorytm wykrywania zakleszczenia
   2. Metody wychodzenia z zakleszczenia
5. Łączenie metod postępowania z zakleszczeniami
6. Algorytm strusia

Plan wykładu

1. Zakleszczenia
   1. Definicja
   2. Dostęp do zasobów współdzielonych
   3. Warunki konieczne do wystąpienia zakleszczenia
   4. Metody opisu zakleszczeń
2. Zapobieganie zakleszczeniom
   1. Wzajemne wykluczanie
   2. Przetrzymywanie i oczekiwanie
   3. Brak wywłaszczeń
   4. Czekanie cykliczne
3. Unikanie zakleszczeń
   1. Algorytm bankiera
   2. Algorytm bezpieczeństwa
   3. Metoda dla zasobów reprezentowanych pojedynczo
4. Wykrywanie i wychodzenie z zakleszczeń
   1. Algorytm wykrywania zakleszczenia
   2. Metody wychodzenia z zakleszczenia
5. Łączenie metod postępowania z zakleszczeniami
6. Algorytm strusia

Plan wykładu

1. Zakleszczenia
   1. Definicja
   2. Dostęp do zasobów współdzielonych
   3. Warunki konieczne do wystąpienia zakleszczenia
   4. Metody opisu zakleszczeń
2. Zapobieganie zakleszczeniom
   1. Wzajemne wykluczanie
   2. Przetrzymywanie i oczekiwanie
   3. Brak wywłaszczeń
   4. Czekanie cykliczne
3. Unikanie zakleszczeń
   1. Algorytm bankiera
   2. Algorytm bezpieczeństwa
   3. Metoda dla zasobów reprezentowanych pojedynczo
4. Wykrywanie i wychodzenie z zakleszczeń
   1. Algorytm wykrywania zakleszczenia
   2. Metody wychodzenia z zakleszczenia
5. Łączenie metod postępowania z zakleszczeniami
6. Algorytm strusia

Plan wykładu

1. Zakleszczenia
   1. Definicja
   2. Dostęp do zasobów współdzielonych
   3. Warunki konieczne do wystąpienia zakleszczenia
   4. Metody opisu zakleszczeń
2. Zapobieganie zakleszczeniom
   1. Wzajemne wykluczanie
   2. Przetrzymywanie i oczekiwanie
   3. Brak wywłaszczeń
   4. Czekanie cykliczne
3. Unikanie zakleszczeń
   1. Algorytm bankiera
   2. Algorytm bezpieczeństwa
   3. Metoda dla zasobów reprezentowanych pojedynczo
4. Wykrywanie i wychodzenie z zakleszczeń
   1. Algorytm wykrywania zakleszczenia
   2. Metody wychodzenia z zakleszczenia
5. Łączenie metod postępowania z zakleszczeniami
6. Algorytm strusia

Plan wykładu

1. Zakleszczenia
   1. Definicja
   2. Dostęp do zasobów współdzielonych
   3. Warunki konieczne do wystąpienia zakleszczenia
   4. Metody opisu zakleszczeń
2. Zapobieganie zakleszczeniom
   1. Wzajemne wykluczanie
   2. Przetrzymywanie i oczekiwanie
   3. Brak wywłaszczeń
   4. Czekanie cykliczne
3. Unikanie zakleszczeń
   1. Algorytm bankiera
   2. Algorytm bezpieczeństwa
   3. Metoda dla zasobów reprezentowanych pojedynczo
4. Wykrywanie i wychodzenie z zakleszczeń
   1. Algorytm wykrywania zakleszczenia
   2. Metody wychodzenia z zakleszczenia
5. Łączenie metod postępowania z zakleszczeniami
6. Algorytm strusia

Plan wykładu

1. Zakleszczenia
   1. Definicja
   2. Dostęp do zasobów współdzielonych
   3. Warunki konieczne do wystąpienia zakleszczenia
   4. Metody opisu zakleszczeń
2. Zapobieganie zakleszczeniom
   1. Wzajemne wykluczanie
   2. Przetrzymywanie i oczekiwanie
   3. Brak wywłaszczeń
   4. Czekanie cykliczne
3. Unikanie zakleszczeń
   1. Algorytm bankiera
   2. Algorytm bezpieczeństwa
   3. Metoda dla zasobów reprezentowanych pojedynczo
4. Wykrywanie i wychodzenie z zakleszczeń
   1. Algorytm wykrywania zakleszczenia
   2. Metody wychodzenia z zakleszczenia
5. Łączenie metod postępowania z zakleszczeniami
6. Algorytm strusia

Plan wykładu

1. Zakleszczenia
   1. Definicja
   2. Dostęp do zasobów współdzielonych
   3. Warunki konieczne do wystąpienia zakleszczenia
   4. Metody opisu zakleszczeń
2. Zapobieganie zakleszczeniom
   1. Wzajemne wykluczanie
   2. Przetrzymywanie i oczekiwanie
   3. Brak wywłaszczeń
   4. Czekanie cykliczne
3. Unikanie zakleszczeń
   1. Algorytm bankiera
   2. Algorytm bezpieczeństwa
   3. Metoda dla zasobów reprezentowanych pojedynczo
4. Wykrywanie i wychodzenie z zakleszczeń
   1. Algorytm wykrywania zakleszczenia
   2. Metody wychodzenia z zakleszczenia
5. Łączenie metod postępowania z zakleszczeniami
6. Algorytm strusia

Plan wykładu

1. Zakleszczenia
   1. Definicja
   2. Dostęp do zasobów współdzielonych
   3. Warunki konieczne do wystąpienia zakleszczenia
   4. Metody opisu zakleszczeń
2. Zapobieganie zakleszczeniom
   1. Wzajemne wykluczanie
   2. Przetrzymywanie i oczekiwanie
   3. Brak wywłaszczeń
   4. Czekanie cykliczne
3. Unikanie zakleszczeń
   1. Algorytm bankiera
   2. Algorytm bezpieczeństwa
   3. Metoda dla zasobów reprezentowanych pojedynczo
4. Wykrywanie i wychodzenie z zakleszczeń
   1. Algorytm wykrywania zakleszczenia
   2. Metody wychodzenia z zakleszczenia
5. Łączenie metod postępowania z zakleszczeniami
6. Algorytm strusia

Plan wykładu

1. Zakleszczenia
   1. Definicja
   2. Dostęp do zasobów współdzielonych
   3. Warunki konieczne do wystąpienia zakleszczenia
   4. Metody opisu zakleszczeń
2. Zapobieganie zakleszczeniom
   1. Wzajemne wykluczanie
   2. Przetrzymywanie i oczekiwanie
   3. Brak wywłaszczeń
   4. Czekanie cykliczne
3. Unikanie zakleszczeń
   1. Algorytm bankiera
   2. Algorytm bezpieczeństwa
   3. Metoda dla zasobów reprezentowanych pojedynczo
4. Wykrywanie i wychodzenie z zakleszczeń
   1. Algorytm wykrywania zakleszczenia
   2. Metody wychodzenia z zakleszczenia
5. Łączenie metod postępowania z zakleszczeniami
6. Algorytm strusia

Plan wykładu

1. Zakleszczenia
   1. Definicja
   2. Dostęp do zasobów współdzielonych
   3. Warunki konieczne do wystąpienia zakleszczenia
   4. Metody opisu zakleszczeń
2. Zapobieganie zakleszczeniom
   1. Wzajemne wykluczanie
   2. Przetrzymywanie i oczekiwanie
   3. Brak wywłaszczeń
   4. Czekanie cykliczne
3. Unikanie zakleszczeń
   1. Algorytm bankiera
   2. Algorytm bezpieczeństwa
   3. Metoda dla zasobów reprezentowanych pojedynczo
4. Wykrywanie i wychodzenie z zakleszczeń
   1. Algorytm wykrywania zakleszczenia
   2. Metody wychodzenia z zakleszczenia
5. Łączenie metod postępowania z zakleszczeniami
6. Algorytm strusia

Plan wykładu

1. Zakleszczenia
   1. Definicja
   2. Dostęp do zasobów współdzielonych
   3. Warunki konieczne do wystąpienia zakleszczenia
   4. Metody opisu zakleszczeń
2. Zapobieganie zakleszczeniom
   1. Wzajemne wykluczanie
   2. Przetrzymywanie i oczekiwanie
   3. Brak wywłaszczeń
   4. Czekanie cykliczne
3. Unikanie zakleszczeń
   1. Algorytm bankiera
   2. Algorytm bezpieczeństwa
   3. Metoda dla zasobów reprezentowanych pojedynczo
4. Wykrywanie i wychodzenie z zakleszczeń
   1. Algorytm wykrywania zakleszczenia
   2. Metody wychodzenia z zakleszczenia
5. Łączenie metod postępowania z zakleszczeniami
6. Algorytm strusia

Plan wykładu

1. Zakleszczenia
   1. Definicja
   2. Dostęp do zasobów współdzielonych
   3. Warunki konieczne do wystąpienia zakleszczenia
   4. Metody opisu zakleszczeń
2. Zapobieganie zakleszczeniom
   1. Wzajemne wykluczanie
   2. Przetrzymywanie i oczekiwanie
   3. Brak wywłaszczeń
   4. Czekanie cykliczne
3. Unikanie zakleszczeń
   1. Algorytm bankiera
   2. Algorytm bezpieczeństwa
   3. Metoda dla zasobów reprezentowanych pojedynczo
4. Wykrywanie i wychodzenie z zakleszczeń
   1. Algorytm wykrywania zakleszczenia
   2. Metody wychodzenia z zakleszczenia
5. Łączenie metod postępowania z zakleszczeniami
6. Algorytm strusia

Definicja

Motto

„Jeśli dwa pociągi zbliżają się do siebie krzyżując swe tory, to oba powinny się całkowicie zatrzymać i nie ruszać ponownie, do czasu, aż drugi z nich odjedzie."
prawo uchwalone przez legislaturę stanu Kansas (USA)

Jeżeli grupa procesów oczekuje na zdarzenie, które może być spowodowane jedynie przez któryś z tych procesów, to taką sytuację nazywamy *zakleszczeniem*[1] lub *impasem (ang. deadlock)*. Przez grupę rozumiemy dwa lub większą liczbę procesów. Do zjawiska zakleszczenia może zazwyczaj dojść przy próbie dostępu do zasobów współdzielonych. Istnieją również inne zjawiska, które mają skutki podobne do zakleszczenia, ale nie spełniają definicji zakleszczeń. Jednym z nich jest *autozakleszczenie*, do którego może dojść, jeśli proces na skutek błędu logicznego będzie próbował zająć zasób, który wcześniej już zajął. Innym przykładem jest *uwięzienie* (ang. livelock), kiedy dwie grupy procesów wykonują przeciwstawne operacje na wspólnych zasobach, co prowadzi do sytuacji, w której mimo tego, że procesy pracują, to ich praca nie wykazuje postępu.

[1] W literaturze polskiej na określenie tego zjawiska był również używany termin „blokada", obecnie już zarzucony.

Dostęp do zasobów współdzielonych

W każdym systemie komputerowym występuje skończona liczba zasobów, które są udostępniane rywalizującym o nie procesom. Wszystkie zasoby można pogrupować, w zależności od ich typów. Zakłada się, że jeśli proces żąda jednego zasobu danego typu, to przydzielenie dowolnego egzemplarza z grupy takich zasobów powinno spełnić to zamówienie. Dostęp do zasobu współdzielonego składa się z trzech etapów:

1. **Zamówienie** Jeśli zasób jest dostępny, to proces otrzymuje go natychmiast, jeśli nie musi poczekać na jego zwolnienie.
2. **Użycie** Proces korzysta z zasobu.
3. **Zwolnienie** Proces oddaje przydzielony zasób.

Wszystkie te etapy (poza być może drugim) wykonywane są za pośrednictwem wywołań systemowych, co pozwala systemowi operacyjnemu na określenie które zasoby są wolne, które zajęte i przez kogo.

Dostęp do zasobów współdzielonych

W każdym systemie komputerowym występuje skończona liczba zasobów, które są udostępniane rywalizującym o nie procesom. Wszystkie zasoby można pogrupować, w zależności od ich typów. Zakłada się, że jeśli proces żąda jednego zasobu danego typu, to przydzielenie dowolnego egzemplarza z grupy takich zasobów powinno spełnić to zamówienie. Dostęp do zasobu współdzielonego składa się z trzech etapów:

1. **Zamówienie** Jeśli zasób jest dostępny, to proces otrzymuje go natychmiast, jeśli nie musi poczekać na jego zwolnienie.
2. **Użycie** Proces korzysta z zasobu.
3. **Zwolnienie** Proces oddaje przydzielony zasób.

Wszystkie te etapy (poza być może drugim) wykonywane są za pośrednictwem wywołań systemowych, co pozwala systemowi operacyjnemu na określenie które zasoby są wolne, które zajęte i przez kogo.

Dostęp do zasobów współdzielonych

W każdym systemie komputerowym występuje skończona liczba zasobów, które są udostępniane rywalizującym o nie procesom. Wszystkie zasoby można pogrupować, w zależności od ich typów. Zakłada się, że jeśli proces żąda jednego zasobu danego typu, to przydzielenie dowolnego egzemplarza z grupy takich zasobów powinno spełnić to zamówienie. Dostęp do zasobu współdzielonego składa się z trzech etapów:

1. **Zamówienie** Jeśli zasób jest dostępny, to proces otrzymuje go natychmiast, jeśli nie musi poczekać na jego zwolnienie.
2. **Użycie** Proces korzysta z zasobu.
3. **Zwolnienie** Proces oddaje przydzielony zasób.

Wszystkie te etapy (poza być może drugim) wykonywane są za pośrednictwem wywołań systemowych, co pozwala systemowi operacyjnemu na określenie które zasoby są wolne, które zajęte i przez kogo.

Dostęp do zasobów współdzielonych

W każdym systemie komputerowym występuje skończona liczba zasobów, które są udostępniane rywalizującym o nie procesom. Wszystkie zasoby można pogrupować, w zależności od ich typów. Zakłada się, że jeśli proces żąda jednego zasobu danego typu, to przydzielenie dowolnego egzemplarza z grupy takich zasobów powinno spełnić to zamówienie. Dostęp do zasobu współdzielonego składa się z trzech etapów:

1. **Zamówienie** Jeśli zasób jest dostępny, to proces otrzymuje go natychmiast, jeśli nie musi poczekać na jego zwolnienie.
2. **Użycie** Proces korzysta z zasobu.
3. **Zwolnienie** Proces oddaje przydzielony zasób.

Wszystkie te etapy (poza być może drugim) wykonywane są za pośrednictwem wywołań systemowych, co pozwala systemowi operacyjnemu na określenie które zasoby są wolne, które zajęte i przez kogo.

Warunki konieczne do wystąpienia zakleszczenia

Istnieją cztery warunki konieczne do powstania zakleszczenia:

1. **Wzajemne wykluczanie** Przynajmniej jeden zasób w systemie musi być niepodzielny, co oznacza, że tego zasobu może używać w określonym czasie tylko jeden proces. Inne procesy, które chcą z niego skorzystać muszą oczekiwać na jego zwolnienie.
2. **Przetrzymywanie i oczekiwanie** Musi istnieć proces, który ma przydzielony co najmniej jeden zasób i równocześnie oczekuje na przydział innego zasobu posiadanego przez inny proces.
3. **Brak wywłaszczeń** Jeśli proces otrzymał zasoby, to nie można mu ich odebrać. Trzeba poczekać, aż zwolni je z własnej inicjatywy.
4. **Czekanie cykliczne** Musi istnieć zbiór procesów $\{P_0, P_1, \ldots, P_n\}$ w stanie oczekiwania, takich że każdy z nich czeka na zasób lub zasoby przetrzymywane przez swojego następnika, a proces P_n czeka na zasób(-oby) przetrzymywany przez P_0.

Warunki konieczne do wystąpienia zakleszczenia

Istnieją cztery warunki konieczne do powstania zakleszczenia:

1. **Wzajemne wykluczanie** Przynajmniej jeden zasób w systemie musi być niepodzielny, co oznacza, że tego zasobu może używać w określonym czasie tylko jeden proces. Inne procesy, które chcą z niego skorzystać muszą oczekiwać na jego zwolnienie.
2. **Przetrzymywanie i oczekiwanie** Musi istnieć proces, który ma przydzielony co najmniej jeden zasób i równocześnie oczekuje na przydział innego zasobu posiadanego przez inny proces.
3. **Brak wywłaszczeń** Jeśli proces otrzymał zasoby, to nie można mu ich odebrać. Trzeba poczekać, aż zwolni je z własnej inicjatywy.
4. **Czekanie cykliczne** Musi istnieć zbiór procesów $\{P_0, P_1, \ldots, P_n\}$ w stanie oczekiwania, takich że każdy z nich czeka na zasób lub zasoby przetrzymywane przez swojego następnika, a proces P_n czeka na zasób(-oby) przetrzymywany przez P_0.

Warunki konieczne do wystąpienia zakleszczenia

Istnieją cztery warunki konieczne do powstania zakleszczenia:

1. **Wzajemne wykluczanie** Przynajmniej jeden zasób w systemie musi być niepodzielny, co oznacza, że tego zasobu może używać w określonym czasie tylko jeden proces. Inne procesy, które chcą z niego skorzystać muszą oczekiwać na jego zwolnienie.
2. **Przetrzymywanie i oczekiwanie** Musi istnieć proces, który ma przydzielony co najmniej jeden zasób i równocześnie oczekuje na przydział innego zasobu posiadanego przez inny proces.
3. **Brak wywłaszczeń** Jeśli proces otrzymał zasoby, to nie można mu ich odebrać. Trzeba poczekać, aż zwolni je z własnej inicjatywy.
4. **Czekanie cykliczne** Musi istnieć zbiór procesów $\{P_0, P_1, \ldots, P_n\}$ w stanie oczekiwania, takich że każdy z nich czeka na zasób lub zasoby przetrzymywane przez swojego następnika, a proces P_n czeka na zasób(-oby) przetrzymywany przez P_0.

Warunki konieczne do wystąpienia zakleszczenia

Istnieją cztery warunki konieczne do powstania zakleszczenia:

1. **Wzajemne wykluczanie** Przynajmniej jeden zasób w systemie musi być niepodzielny, co oznacza, że tego zasobu może używać w określonym czasie tylko jeden proces. Inne procesy, które chcą z niego skorzystać muszą oczekiwać na jego zwolnienie.
2. **Przetrzymywanie i oczekiwanie** Musi istnieć proces, który ma przydzielony co najmniej jeden zasób i równocześnie oczekuje na przydział innego zasobu posiadanego przez inny proces.
3. **Brak wywłaszczeń** Jeśli proces otrzymał zasoby, to nie można mu ich odebrać. Trzeba poczekać, aż zwolni je z własnej inicjatywy.
4. **Czekanie cykliczne** Musi istnieć zbiór procesów $\{P_0, P_1, \ldots, P_n\}$ w stanie oczekiwania, takich że każdy z nich czeka na zasób lub zasoby przetrzymywane przez swojego następnika, a proces P_n czeka na zasób(-oby) przetrzymywany przez P_0.

Warunki konieczne do wystąpienia zakleszczenia

Istnieją cztery warunki konieczne do powstania zakleszczenia:

1. **Wzajemne wykluczanie** Przynajmniej jeden zasób w systemie musi być niepodzielny, co oznacza, że tego zasobu może używać w określonym czasie tylko jeden proces. Inne procesy, które chcą z niego skorzystać muszą oczekiwać na jego zwolnienie.
2. **Przetrzymywanie i oczekiwanie** Musi istnieć proces, który ma przydzielony co najmniej jeden zasób i równocześnie oczekuje na przydział innego zasobu posiadanego przez inny proces.
3. **Brak wywłaszczeń** Jeśli proces otrzymał zasoby, to nie można mu ich odebrać. Trzeba poczekać, aż zwolni je z własnej inicjatywy.
4. **Czekanie cykliczne** Musi istnieć zbiór procesów $\{P_0, P_1, \ldots, P_n\}$ w stanie oczekiwania, takich że każdy z nich czeka na zasób lub zasoby przetrzymywane przez swojego następnika, a proces P_n czeka na zasób(-oby) przetrzymywany przez P_0.

Warunki konieczne do wystąpienia zakleszczenia

Istnieją cztery warunki konieczne do powstania zakleszczenia:

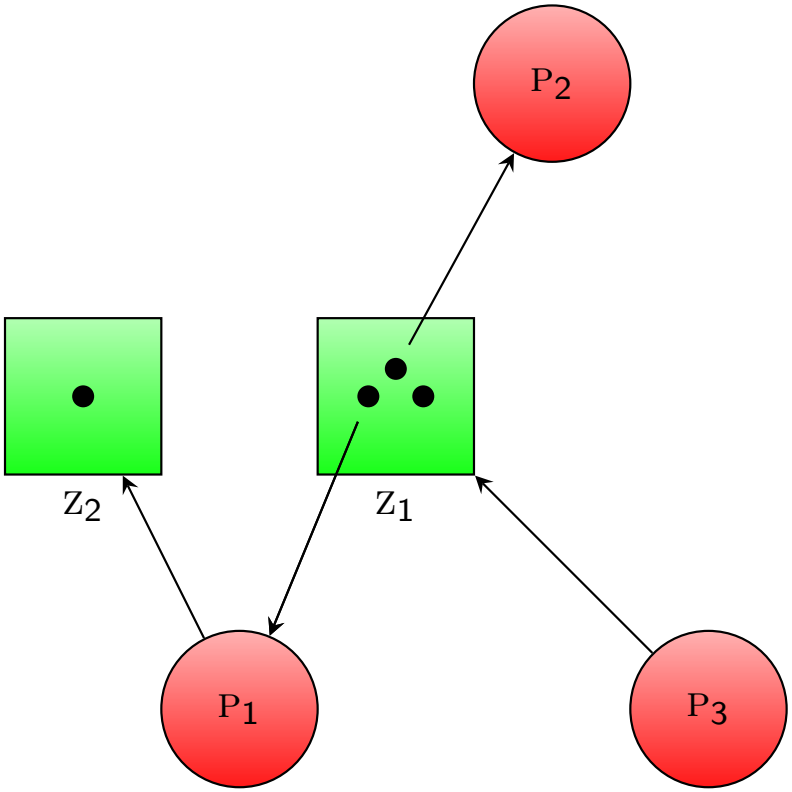
1. **Wzajemne wykluczanie** Przynajmniej jeden zasób w systemie musi być niepodzielny, co oznacza, że tego zasobu może używać w określonym czasie tylko jeden proces. Inne procesy, które chcą z niego skorzystać muszą oczekiwać na jego zwolnienie.
2. **Przetrzymywanie i oczekiwanie** Musi istnieć proces, który ma przydzielony co najmniej jeden zasób i równocześnie oczekuje na przydział innego zasobu posiadanego przez inny proces.
3. **Brak wywłaszczeń** Jeśli proces otrzymał zasoby, to nie można mu ich odebrać. Trzeba poczekać, aż zwolni je z własnej inicjatywy.
4. **Czekanie cykliczne** Musi istnieć zbiór procesów $\{P_0, P_1, \ldots, P_n\}$ w stanie oczekiwania, takich że każdy z nich czeka na zasób lub zasoby przetrzymywane przez swojego następnika, a proces P_n czeka na zasób(-oby) przetrzymywany przez P_0.

Aby wystąpiło zakleszczenie, wszystkie powyższe warunki muszą zostać spełnione.

Metody opisu zakleszczeń

Za pomocą *grafu przydziału zasobów systemu* możemy opisać zjawisko występowania zakleszczeń. Taki graf jest grafem skierowanym, składającym się ze zbioru krawędzi $\mathbb{K}$ i zbioru wierzchołków $\mathbb{W}$. Zbiór wierzchołków dzieli się na dwa podzbiory: zbiór $\mathbb{P}$ wszystkich procesów systemu, oraz zbiór $\mathbb{Z}$ wszystkich typów zasobów w systemie. Krawędź skierowaną od procesu P_i do zasobu Z_j ($P_i \rightarrow Z_j$) nazywamy *krawędzią zamówienia*. Krawędź skierowaną odwrotnie ($P_i \leftarrow Z_j$) nazywamy *krawędzią przydziału*. Na następnych trzech planszach znajdują się grafy przydziału zasobów, które modelują różne scenariusze związane z przydziałem zasobów.

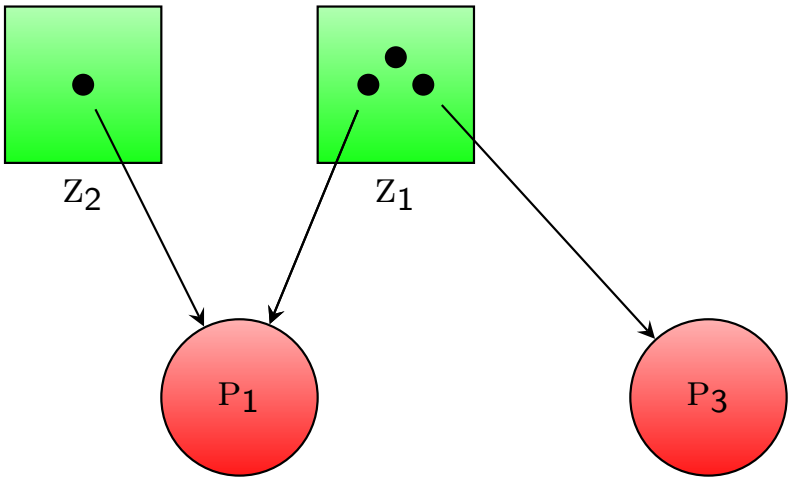
Graf acykliczny — brak zakleszczeń



Rysunek: Graf przydziału zasobów systemowych bez cykli

Przedstawiony po lewej stronie graf przydziału zasobów składa się z trzech wierzchołków P i dwóch wierzchołków Z. Punkty wewnątrz wierzchołków Z oznaczają pojedyncze egzemplarze zasobów danego typu. Należy zwrócić uwagę, że krawędzie zamówienia biegną od wierzchołka procesu do wierzchołka typu zasobu, natomiast krawędzie przydziału biegną od konkretnego egzemplarza zasobu, do wierzchołka procesu, który go otrzymał. Ponieważ w grafie nie ma cyklu, to żadne procesy nie uległy zakleszczeniu.

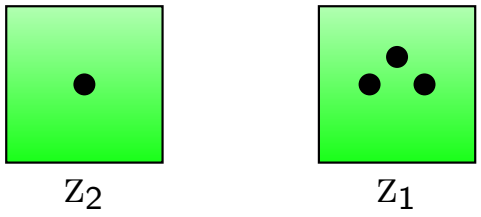
Graf acykliczny — brak zakleszczeń



Rysunek: Graf przydziału zasobów systemowych bez cykli

Przedstawiony po lewej stronie graf przydziału zasobów składa się z trzech wierzchołków P i dwóch wierzchołków Z. Punkty wewnątrz wierzchołków Z oznaczają pojedyncze egzemplarze zasobów danego typu. Należy zwrócić uwagę, że krawędzie zamówienia biegną od wierzchołka procesu do wierzchołka typu zasobu, natomiast krawędzie przydziału biegną od konkretnego egzemplarza zasobu, do wierzchołka procesu, który go otrzymał. Ponieważ w grafie nie ma cyklu, to żadne procesy nie uległy zakleszczeniu.

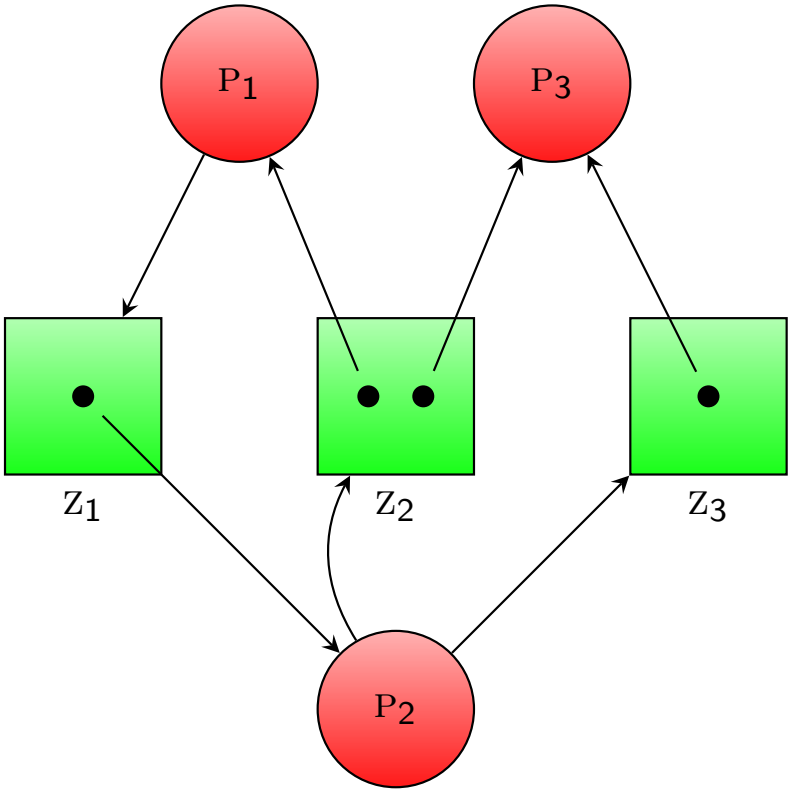
Graf acykliczny — brak zakleszczeń



Rysunek: Graf przydziału zasobów systemowych bez cykli

Przedstawiony po lewej stronie graf przydziału zasobów składa się z trzech wierzchołków P i dwóch wierzchołków Z. Punkty wewnątrz wierzchołków Z oznaczają pojedyncze egzemplarze zasobów danego typu. Należy zwrócić uwagę, że krawędzie zamówienia biegną od wierzchołka procesu do wierzchołka typu zasobu, natomiast krawędzie przydziału biegną od konkretnego egzemplarza zasobu, do wierzchołka procesu, który go otrzymał. Ponieważ w grafie nie ma cyklu, to żadne procesy nie uległy zakleszczeniu.

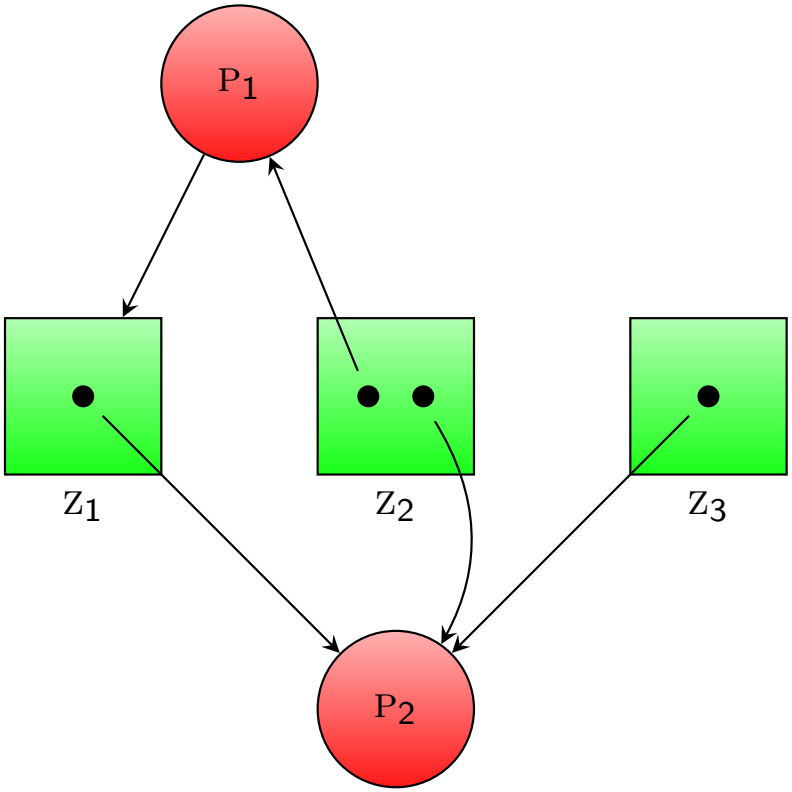
Graf cykliczny — brak zakleszczeń



Rysunek: Graf przydziału zasobów systemowych z cyklem, ale bez zakleszczenia.

W przedstawionym obok grafie przydziału zasobów występuje cykl ($P_1 \rightarrow Z_1 \rightarrow P_2 \rightarrow Z_2 \rightarrow P_1$), mimo to nie dochodzi do zakleszczenia.

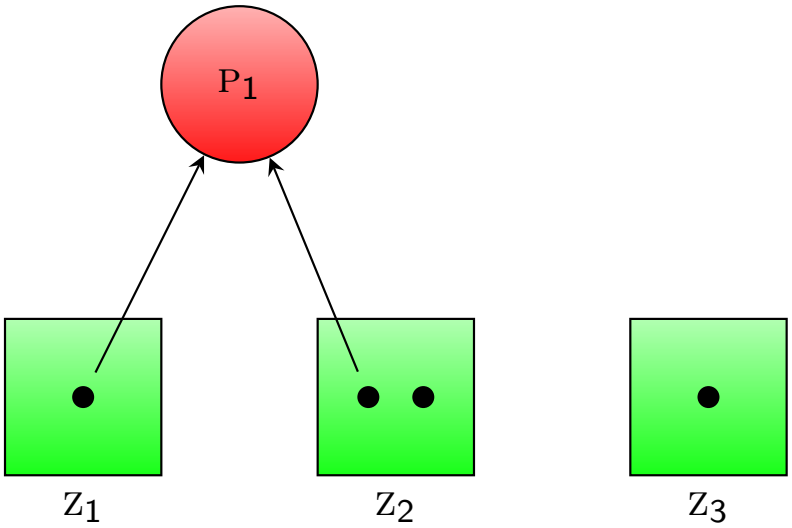
Graf cykliczny — brak zakleszczeń



Rysunek: Graf przydziału zasobów systemowych z cyklem, ale bez zakleszczenia.

W przedstawionym obok grafie przydziału zasobów występuje cykl ($P_1 \rightarrow Z_1 \rightarrow P_2 \rightarrow Z_2 \rightarrow P_1$), mimo to nie dochodzi do zakleszczenia.

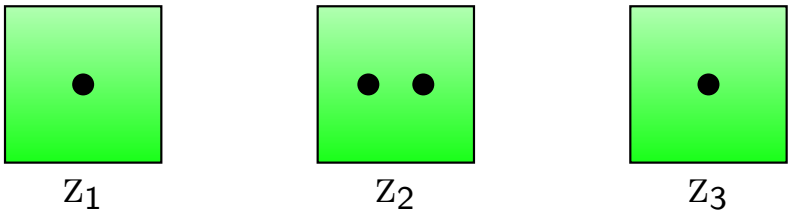
Graf cykliczny — brak zakleszczeń



Rysunek: Graf przydziału zasobów systemowych z cyklem, ale bez zakleszczenia.

W przedstawionym obok grafie przydziału zasobów występuje cykl ($P_1 \rightarrow Z_1 \rightarrow P_2 \rightarrow Z_2 \rightarrow P_1$), mimo to nie dochodzi do zakleszczenia.

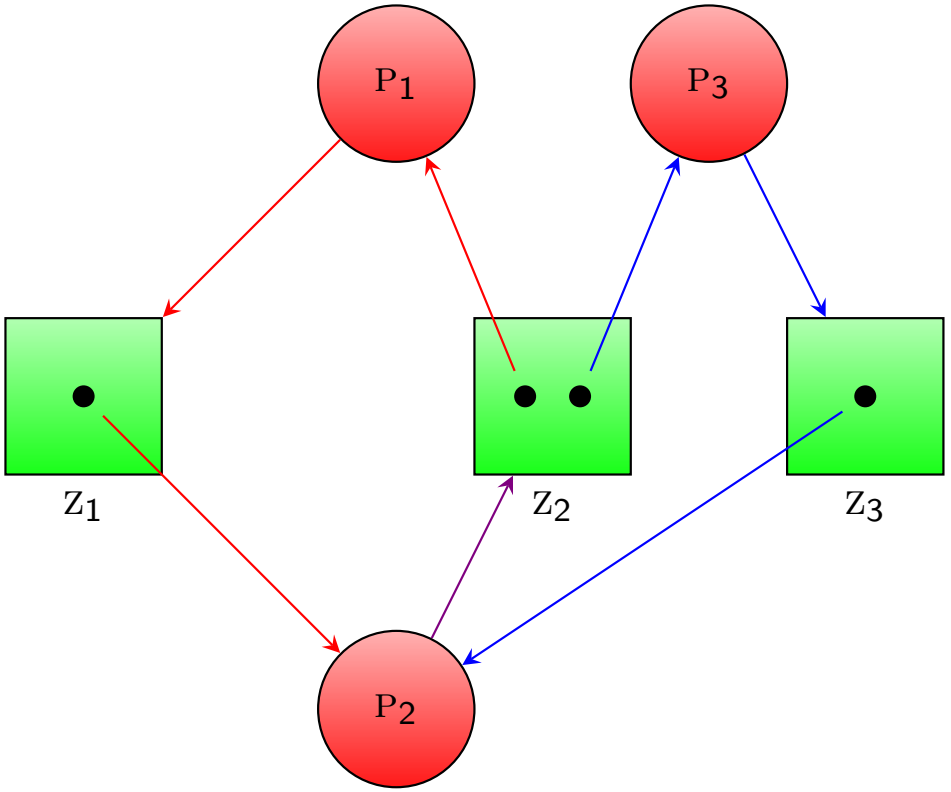
Graf cykliczny — brak zakleszczeń



Rysunek: Graf przydziału zasobów systemowych z cyklem, ale bez zakleszczenia.

W przedstawionym obok grafie przydziału zasobów występuje cykl ($P_1 \rightarrow Z_1 \rightarrow P_2 \rightarrow Z_2 \rightarrow P_1$), mimo to nie dochodzi do zakleszczenia.

Graf cykliczny — zakleszczenie



Rysunek: Graf przydziału zasobów systemowych z cyklem i zakleszczeniem.

W grafie po lewej stronie występują dwa cykle minimalne:
$P_1 \rightarrow Z_1 \rightarrow P_2 \rightarrow Z_2$
oraz
$P_2 \rightarrow Z_2 \rightarrow P_3 \rightarrow Z_3 \rightarrow P_2$.
W tym grafie dochodzi do zakleszczenia.

Grafy przydziału zasobów — podsumowanie

Warunkiem koniecznym do wystąpienia zakleszczenia w systemie jest to, aby w grafie przydziału zasobów powstał cykl. Nie jest to jednak *warunek wystarczający*. Jak pokazuje to drugi z zaprezentowanych rysunków, w grafie może powstać cykl, mimo to nie dojdzie do zakleszczenia.

Zapobieganie zakleszczeniom

Istnieją trzy rodzaje metod radzenia sobie z zakleszczeniami. Nim zapoznamy się z nimi warto zaznaczyć, że większość dzisiejszych systemów operacyjnych nie stosuje żadnych metod zawiązanych z obsługą zakleszczeń, lub stosuje je nie wprost, czyli nie w postaci wbudowanych mechanizmów. Opisy tych metod zaczniemy od metod zapobiegania zakleszczeniom. Aby w systemie komputerowym nigdy nie wystąpiło zakleszczenie, trzeba zadbać, aby *co najmniej jeden* warunek konieczny do wystąpienia zakleszczenia nie był nigdy spełniony. Na kolejnych planszach zostaną przeanalizowane możliwości zapobiegania spełnieniu poszczególnych warunków.

Wzajemne wykluczanie

Zarówno zasoby lokalne procesów, jak i zasoby przeznaczone tylko do odczytu (np.: wybrane przez twórców pliki, port zegara czasu rzeczywistego) nie muszą być objęte warunkiem wzajemnego wykluczania. Inaczej jest w przypadku tzw. *zasobów niepodzielnych*, które mogą być modyfikowane. One muszą być użytkowane na zasadzie wyłączności, czego wymaga problem sekcji krytycznej. Ponieważ takie zasoby zawsze istnieją w systemie komputerowym, to warunek wzajemnego wykluczania jest wręcz niezbędny do poprawnej pracy procesów użytkownika.

Przetrzymywanie i oczekiwanie

Aby zapewnić, że ten warunek nigdy nie będzie spełniony należy wymusić na procesach, aby zamawiały zasoby, tylko wtedy, gdy nie są w posiadaniu innych zasobów. Istnieją dwa sposoby na osiągnięcie tego celu. Pierwszy polega na tym, że każdy proces musi zamawiać wszystkie niezbędne do pracy zasoby zaraz po uruchomieniu. Wywołania wszystkich niezbędnych funkcji systemowych (wywołań) związanych z zamawianiem zasobów muszą w programach poprzedzać wszelkie inne operacje. Drugi sposób nakazuje procesowi przed zamówieniem nowego zasobu oddanie wszystkich, które posiada. Oba te sposoby mogą prowadzić do zmniejszenia stopnia wykorzystania zasobów współdzielonych i/lub do głodzenia procesów.

Brak wywłaszczeń

Brak spełnienia tego warunku można również zapewnić na dwa sposoby, poprzez określenie odpowiedniego protokołu zamawiania zasobów. W pierwszym rozwiązaniu, jeśli proces zamawiający zasób musi czekać na jego przydział, to system operacyjny niejawnie odbiera mu wszystkie zasoby jakie są w jego posiadaniu i niejawnie dopisuje je do listy jego zamówień. Proces jest budzony z oczekiwania dopiero wtedy, gdy można przydzielić mu wszystkie te zasoby. Drugi sposób zakłada, że jeśli proces zamawia zasób, który jest w posiadaniu innego procesu, który czeka na przydział innych zasobów, to temu ostatniemu procesowi zasób jest odbierany i przydzielany pierwszemu. Ostatni protokół nie nadaje się dla zasobów, których stanu nie można łatwo skopiować i w przyszłości odtworzyć.

Czekanie cykliczne

Można zagwarantować niespełnienie warunku czekania cyklicznego nadając każdemu typowi zasobów określony, unikatowy numer porządkowy (liczbę naturalną) i wymuszając na procesach zamawianie zasobów, według rosnącej numeracji. Oznacza to, że proces, który ma już w swoim posiadaniu zasób nr 4, może zająć zasób nr 5, ale już nie może zająć zasobu nr 3, a nawet kolejnego egzemplarza zasobu nr 4. Alternatywnie można wymagać, aby proces posiadający zasób nr 4 zwolnił go przed zamówieniem zasobu nr 3. Na zasadzie konwencji ta metoda jest stosowana w jądrze Linuksa, tzn. programiści przestrzegają, aby np.: semafory były zajmowane i zwalniane w określonej kolejności.

Unikanie zakleszczeń

Ponieważ zapobieganie zakleszczeniom polega na ograniczaniu przydziału zasobów procesom, to jego niekorzystnym skutkiem ubocznym może być zmniejszenie wykorzystania tych zasobów, co z kolei prowadzi do obniżenia przepustowości systemu. Istnieją jednak inne metody, które można stosować zamiast zapobiegania zakleszczeniom. Jedną z tych metod jest *unikanie zakleszczeń*. Analizując grafy przydziału zasobów, można zauważyć, że system może znaleźć się w trzech stanach: *stanie bezpiecznym*, *stanie zagrożonym* i *stanie zakleszczenia*. Należy podkreślić, że stan zagrożenia nie zawsze jest stanem zakleszczenia, ale stan zakleszczenia jest zawsze stanem zagrożenia. Unikanie zakleszczeń pozwala utrzymywać system w stanie bezpiecznym. Aby to osiągnąć system operacyjny musi wiedzieć jak będzie następowało zamawianie zasobów przez poszczególne procesy. Mając te informacje może on dynamicznie określać, czy dany ciąg zamówień dokonywanych przez procesy jest *ciągiem bezpiecznym*, czy nie prowadzi do cyklicznego oczekiwania. W wyniku stosowania tej metody proces może nie otrzymać zasobu, mimo że jest on wolny, jeśli realizacja tego zamówienia prowadziłaby do stanu zagrożenia. Jednym z algorytmów pozwalających na unikanie zakleszczeń jest *algorytm bankiera* przedstawiony na następnych planszach.

Struktury danych wymagane przez algorytm bankiera

Załóżmy, że w systemie mamy n procesów i m typów zasobów (obie te liczby mogą ulegać zmianie wraz z biegiem czasu). Algorytm bankiera wymaga następujących struktur danych:

- **Dostępne** - tablica (wektor) o m elementach, określająca liczbę dostępnych egzemplarzy zasobów każdego typu. Jeśli **Dostępne**[j] $==$ x, to oznacza, że dostępnych jest x egzemplarzy zasobu typu j.
- **Maksymalne** - macierz, o wymiarach $n \times m$, która określa maksymalne zapotrzebowania procesów na zasoby poszczególnych typów. Jeśli **Maksymalne**[i][j] $==$ x, to oznacza, że proces P_i może zamówić maksymalnie x egzemplarzy zasobu typu j.
- **Przydzielone** - macierz o wymiarach $n \times m$, określa liczbę zasobów poszczególnych typów, które już zostały przydzielone procesom.
- **Potrzebne** - macierz rozmiaru $n \times m$ określająca liczbę wolnych zasobów poszczególnych typów potrzebną do zaspokojenia maksymalnego zapotrzebowania procesów. Należy zauważyć, że **Potrzebne**[i][j]$==$**Maksymalne**[i][j]-**Przydzielone**[i][j].
- **Zamówienia**$_i$ - tablica o m elementach opisująca pojedyncze zamówienie procesu P_i na egzemplarze każdego z rodzajów zasobów.

Aby algorytm działał poprawnie, należy zdefiniować również operator porównania dwóch wektorów: $X \leqslant Y \Leftrightarrow \forall i = 1, 2, \ldots, n, X[i] \leqslant Y[i]$

Struktury danych wymagane przez algorytm bankiera

Załóżmy, że w systemie mamy n procesów i m typów zasobów (obie te liczby mogą ulegać zmianie wraz z biegiem czasu). Algorytm bankiera wymaga następujących struktur danych:

- **Dostępne** - tablica (wektor) o m elementach, określająca liczbę dostępnych egzemplarzy zasobów każdego typu. Jeśli **Dostępne**[j] $==$ x, to oznacza, że dostępnych jest x egzemplarzy zasobu typu j.
- **Maksymalne** - macierz, o wymiarach $n \times m$, która określa maksymalne zapotrzebowania procesów na zasoby poszczególnych typów. Jeśli **Maksymalne**[i][j] $==$ x, to oznacza, że proces P_i może zamówić maksymalnie x egzemplarzy zasobu typu j.
- **Przydzielone** - macierz o wymiarach $n \times m$, określa liczbę zasobów poszczególnych typów, które już zostały przydzielone procesom.
- **Potrzebne** - macierz rozmiaru $n \times m$ określająca liczbę wolnych zasobów poszczególnych typów potrzebną do zaspokojenia maksymalnego zapotrzebowania procesów. Należy zauważyć, że **Potrzebne**[i][j]==**Maksymalne**[i][j]-**Przydzielone**[i][j].
- **Zamówienia**$_i$ - tablica o m elementach opisująca pojedyncze zamówienie procesu P_i na egzemplarze każdego z rodzajów zasobów.

Aby algorytm działał poprawnie, należy zdefiniować również operator porównania dwóch wektorów: $X \leqslant Y \Leftrightarrow \forall i = 1, 2, \ldots, n, X[i] \leqslant Y[i]$

Struktury danych wymagane przez algorytm bankiera

Załóżmy, że w systemie mamy n procesów i m typów zasobów (obie te liczby mogą ulegać zmianie wraz z biegiem czasu). Algorytm bankiera wymaga następujących struktur danych:

- **Dostępne** - tablica (wektor) o m elementach, określająca liczbę dostępnych egzemplarzy zasobów każdego typu. Jeśli **Dostępne**[j] $==$ x, to oznacza, że dostępnych jest x egzemplarzy zasobu typu j.
- **Maksymalne** - macierz, o wymiarach $n \times m$, która określa maksymalne zapotrzebowania procesów na zasoby poszczególnych typów. Jeśli **Maksymalne**[i][j] $==$ x, to oznacza, że proces P_i może zamówić maksymalnie x egzemplarzy zasobu typu j.
- **Przydzielone** - macierz o wymiarach $n \times m$, określa liczbę zasobów poszczególnych typów, które już zostały przydzielone procesom.
- **Potrzebne** - macierz rozmiaru $n \times m$ określająca liczbę wolnych zasobów poszczególnych typów potrzebną do zaspokojenia maksymalnego zapotrzebowania procesów. Należy zauważyć, że **Potrzebne**[i][j]==**Maksymalne**[i][j]-**Przydzielone**[i][j].
- **Zamówienia**$_i$ - tablica o m elementach opisująca pojedyncze zamówienie procesu P_i na egzemplarze każdego z rodzajów zasobów.

Aby algorytm działał poprawnie, należy zdefiniować również operator porównania dwóch wektorów: $X \leqslant Y \Leftrightarrow \forall i = 1, 2, \ldots, n, X[i] \leqslant Y[i]$

Struktury danych wymagane przez algorytm bankiera

Załóżmy, że w systemie mamy n procesów i m typów zasobów (obie te liczby mogą ulegać zmianie wraz z biegiem czasu). Algorytm bankiera wymaga następujących struktur danych:

- **Dostępne** - tablica (wektor) o m elementach, określająca liczbę dostępnych egzemplarzy zasobów każdego typu. Jeśli **Dostępne**[j] $==$ x, to oznacza, że dostępnych jest x egzemplarzy zasobu typu j.
- **Maksymalne** - macierz, o wymiarach $n \times m$, która określa maksymalne zapotrzebowania procesów na zasoby poszczególnych typów. Jeśli **Maksymalne**[i][j] $==$ x, to oznacza, że proces P_i może zamówić maksymalnie x egzemplarzy zasobu typu j.
- **Przydzielone** - macierz o wymiarach $n \times m$, określa liczbę zasobów poszczególnych typów, które już zostały przydzielone procesom.
- **Potrzebne** - macierz rozmiaru $n \times m$ określająca liczbę wolnych zasobów poszczególnych typów potrzebną do zaspokojenia maksymalnego zapotrzebowania procesów. Należy zauważyć, że **Potrzebne**[i][j]==**Maksymalne**[i][j]-**Przydzielone**[i][j].
- **Zamówienia**$_i$ - tablica o m elementach opisująca pojedyncze zamówienie procesu P_i na egzemplarze każdego z rodzajów zasobów.

Aby algorytm działał poprawnie, należy zdefiniować również operator porównania dwóch wektorów: $X \leqslant Y \Leftrightarrow \forall i = 1, 2, \ldots, n, X[i] \leqslant Y[i]$

Struktury danych wymagane przez algorytm bankiera

Załóżmy, że w systemie mamy n procesów i m typów zasobów (obie te liczby mogą ulegać zmianie wraz z biegiem czasu). Algorytm bankiera wymaga następujących struktur danych:

- **Dostępne** - tablica (wektor) o m elementach, określająca liczbę dostępnych egzemplarzy zasobów każdego typu. Jeśli **Dostępne**[j] $==$ x, to oznacza, że dostępnych jest x egzemplarzy zasobu typu j.
- **Maksymalne** - macierz, o wymiarach $n \times m$, która określa maksymalne zapotrzebowania procesów na zasoby poszczególnych typów. Jeśli **Maksymalne**[i][j] $==$ x, to oznacza, że proces P_i może zamówić maksymalnie x egzemplarzy zasobu typu j.
- **Przydzielone** - macierz o wymiarach $n \times m$, określa liczbę zasobów poszczególnych typów, które już zostały przydzielone procesom.
- **Potrzebne** - macierz rozmiaru $n \times m$ określająca liczbę wolnych zasobów poszczególnych typów potrzebną do zaspokojenia maksymalnego zapotrzebowania procesów. Należy zauważyć, że **Potrzebne**[i][j]==**Maksymalne**[i][j]-**Przydzielone**[i][j].
- **Zamówienia**$_i$ - tablica o m elementach opisująca pojedyncze zamówienie procesu P_i na egzemplarze każdego z rodzajów zasobów.

Aby algorytm działał poprawnie, należy zdefiniować również operator porównania dwóch wektorów: $X \leqslant Y \Leftrightarrow \forall i = 1, 2, \ldots, n, X[i] \leqslant Y[i]$

Struktury danych wymagane przez algorytm bankiera

Załóżmy, że w systemie mamy n procesów i m typów zasobów (obie te liczby mogą ulegać zmianie wraz z biegiem czasu). Algorytm bankiera wymaga następujących struktur danych:

- **Dostępne** - tablica (wektor) o m elementach, określająca liczbę dostępnych egzemplarzy zasobów każdego typu. Jeśli **Dostępne**[j] $==$ x, to oznacza, że dostępnych jest x egzemplarzy zasobu typu j.
- **Maksymalne** - macierz, o wymiarach $n \times m$, która określa maksymalne zapotrzebowania procesów na zasoby poszczególnych typów. Jeśli **Maksymalne**[i][j] $==$ x, to oznacza, że proces P_i może zamówić maksymalnie x egzemplarzy zasobu typu j.
- **Przydzielone** - macierz o wymiarach $n \times m$, określa liczbę zasobów poszczególnych typów, które już zostały przydzielone procesom.
- **Potrzebne** - macierz rozmiaru $n \times m$ określająca liczbę wolnych zasobów poszczególnych typów potrzebną do zaspokojenia maksymalnego zapotrzebowania procesów. Należy zauważyć, że **Potrzebne**[i][j]==**Maksymalne**[i][j]-**Przydzielone**[i][j].
- **Zamówienia**$_i$ - tablica o m elementach opisująca pojedyncze zamówienie procesu P_i na egzemplarze każdego z rodzajów zasobów.

Aby algorytm działał poprawnie, należy zdefiniować również operator porównania dwóch wektorów: $X \leqslant Y \Leftrightarrow \forall i = 1, 2, \ldots, n, X[i] \leqslant Y[i]$

Algorytm bankiera

Załóżmy, że proces P_i dokonuje zamówienia. Aby sprawdzić, czy może zrealizować to zamówienie system operacyjny wykonuje następujący algorytm (zapis $\mathbf{Przydzielone}_i$ oznacza i-ty wiersz macierzy **Przydzielone**, który traktujemy jak wektor):

1. Jeśli $\mathbf{Zamówiena}_i \leqslant \mathbf{Potrzebne}_i$ to wykonaj krok 2. W przeciwnym przypadku wyrzuć wyjątek - proces przekroczył zgłoszone maksymalne zapotrzebowanie na zasoby.
2. Jeśli $\mathbf{Zamówienia}_i \leqslant \mathbf{Dostępne}$, to wykonaj krok 3. W przeciwnym razie proces P_i musi przejść w stan oczekiwania, bo żądane przez niego zasoby są niedostępne.
3. System podejmuje próbę przydzielenia zasobów procesowi P_i, modyfikując zawartość odpowiednich macierzy:
   $\mathbf{Dostępne} \leftarrow \mathbf{Dostępne} - \mathbf{Zamówienia}_i$;
   $\mathbf{Przydzielone}_i \leftarrow \mathbf{Przydzielone}_i + \mathbf{Zamówienia}_i$;
   $\mathbf{Potrzebne}_i \leftarrow \mathbf{Potrzebne}_i - \mathbf{Zamówienia}_i$;
   Po wykonaniu tej modyfikacji system sprawdza, czy stan wynikowy jest bezpieczny, wykonując *algorytm bezpieczeństwa* opisany na następnej planszy. Jeśli po wykonaniu przedziału system byłby w stanie zagrożenia, to ten przydział jest wstrzymywany.

Algorytm bezpieczeństwa

1. Niech **Praca** i **Koniec** oznaczają wektory o długości odpowiednio m i n. W pierwszym kroku dokonywane są dwa przypisania: **Praca** $\leftarrow$ **Dostępne** oraz **Koniec**[i] $\leftarrow$ *false* dla $i = 1, 2, \ldots, n$.
2. Znajdujemy takie i, że zarówno:
   Koniec[i] $==$ *false*;
   $\mathbf{Potrzebne}_i \leqslant$ **Praca**;
   Jeśli takie i nie istnieje, to wykonujemy krok 4.
3. **Praca** $\leftarrow$ **Praca** $+$ $\mathbf{Przydzielone}_i$;
   Koniec[i] $\leftarrow$ *true*;
   Skok do kroku 2.
4. Jeśli **Koniec**[i] $==$ *true* dla wszystkich możliwych i, to system jest w stanie bezpiecznym.

Unikanie zakleszczeń dla zasobów reprezentowanych pojedynczo

Złożoność algorytmu bankiera wynosi $O(m \times n^2)$. W systemach, w których występuje tylko jeden egzemplarz wszystkich typów zasobów można zastosować prostszy algorytm o mniejszej złożoności. Jest to algorytm grafowy, a graf na którym przeprowadzane są operacje jest zmodyfikowanym grafem przydziału zasobów. Modyfikacja polega na wprowadzeniu nowego rodzaju krawędzi, zwanych *krawędziami deklaracji*. Krawędź deklaracji $P_i \rightarrow Z_j$ określa, że proces P_i będzie chciał w przyszłości zamówić zasób Z_j. Jeśli dojdzie do takiego zamówienia, to krawędź deklaracji zamieniana jest automatycznie w krawędź zamówienia. Dla odróżnienia na rysunku tych dwóch krawędzi, krawędź deklaracji rysujemy przerywaną linią. Po oddaniu przez proces zasobu krawędź przydziału jest zmieniana w krawędź deklaracji. Można wymagać, aby krawędzie deklaracji pojawiały się w grafie wraz z wierzchołkiem procesu, albo żeby nowe krawędzie tego typu były dodawane tylko wtedy, gdy wszystkie krawędzie związane z procesem są krawędziami deklaracji. Do wykrywania zakleszczeń w takim grafie wystarczy zastosować algorytm wykrywania cykli, o złożoności $O(n^2)$.

Wykrywanie i wychodzenie z zakleszczeń

Trzecia metoda radzenia sobie z zakleszczeniami polega na ich wykrywaniu i wychodzeniu z nich. Potrzebne są więc dwa odrębne algorytmy

- Algorytm wykrywania zakleszczenia.
- Algorytm wychodzenia z zakleszczenia.

Analizę tego rozwiązania rozpoczniemy opisem pierwszego z wymienionych algorytmów. Proszę zwrócić uwagę na podobieństwo tego algorytmu do algorytmu bezpieczeństwa.

Struktury danych dla algorytmu wykrywania zakleszczeń

- **Dostępne** - tablica (wektor) o m elementach, określająca liczbę dostępnych egzemplarzy zasobów każdego typu.
- **Przydzielone** - macierz o wymiarach $n \times m$, określa liczbę zasobów poszczególnych typów, które już zostały przydzielone procesom.
- **Zamówienia** - macierz rozmiaru $n \times m$ określająca bieżące zamówienia każdego z procesów.

Struktury danych dla algorytmu wykrywania zakleszczeń

- **Dostępne** - tablica (wektor) o m elementach, określająca liczbę dostępnych egzemplarzy zasobów każdego typu.
- **Przydzielone** - macierz o wymiarach $n \times m$, określa liczbę zasobów poszczególnych typów, które już zostały przydzielone procesom.
- **Zamówienia** - macierz rozmiaru $n \times m$ określająca bieżące zamówienia każdego z procesów.

Struktury danych dla algorytmu wykrywania zakleszczeń

- **Dostępne** - tablica (wektor) o m elementach, określająca liczbę dostępnych egzemplarzy zasobów każdego typu.
- **Przydzielone** - macierz o wymiarach $n \times m$, określa liczbę zasobów poszczególnych typów, które już zostały przydzielone procesom.
- **Zamówienia** - macierz rozmiaru $n \times m$ określająca bieżące zamówienia każdego z procesów.

Struktury danych dla algorytmu wykrywania zakleszczeń

- **Dostępne** - tablica (wektor) o m elementach, określająca liczbę dostępnych egzemplarzy zasobów każdego typu.
- **Przydzielone** - macierz o wymiarach $n \times m$, określa liczbę zasobów poszczególnych typów, które już zostały przydzielone procesom.
- **Zamówienia** - macierz rozmiaru $n \times m$ określająca bieżące zamówienia każdego z procesów.

Struktury danych dla algorytmu wykrywania zakleszczeń

- **Dostępne** - tablica (wektor) o m elementach, określająca liczbę dostępnych egzemplarzy zasobów każdego typu.
- **Przydzielone** - macierz o wymiarach $n \times m$, określa liczbę zasobów poszczególnych typów, które już zostały przydzielone procesom.
- **Zamówienia** - macierz rozmiaru $n \times m$ określająca bieżące zamówienia każdego z procesów.

Dodatkowo przyjmujemy że działanie operatora „mniejszy lub równy” jest takie samo, jak w algorytmie bankiera.

Algorytm wykrywania zakleszczeń

1. Niech **Praca** i **Koniec** oznaczają wektory o długości odpowiednio m i n. W pierwszym kroku dokonywane są dwa przypisania: **Praca** $\leftarrow$ **Dostępne** i jeśli **Przydzielone**$_i \neq 0$ to **Koniec**[i] $\leftarrow$ *false*, w przeciwnym przypadku **Koniec**[i] $\leftarrow$ *true*, dla $i = 1, 2, \ldots, n$.
2. Znajdujemy takie i, że zarówno:
   Koniec[i] == *false*;
   Zamówienia$_i \leqslant$ **Praca**;
   Jeśli takie i nie istnieje, to wykonujemy krok 4.
3. **Praca** $\leftarrow$ **Praca** + **Przydzielone**$_i$;
   Koniec[i] $\leftarrow$ *true*;
   Skok do kroku 2.
4. Jeśli dla pewnych wartości i z przedziału [1,n], **Koniec**[i] == *false*, to system jest w stanie zakleszczenia, dokładniej wszystkie procesy dla których **Koniec**[i] == *false* są w stanie zakleszczenia.

Algorytm wykrywania zakleszczeń — przykład

Przydzielone

	A	B	C
P_0	2	1	0
P_1	1	2	1
P_2	1	1	1

Zamówienia

	A	B	C
P_0	1	2	1
P_1	1	1	1
P_2	0	1	0

Dostępne

A	B	C
0	1	0

Koniec

P_0
P_1
P_2

Praca

A	B	C

Algorytm wykrywania zakleszczeń — przykład

Przydzielone

	A	B	C
P_0	2	1	0
P_1	1	2	1
P_2	1	1	1

Zamówienia

	A	B	C
P_0	1	2	1
P_1	1	1	1
P_2	0	1	0

Dostępne

A	B	C
0	1	0

Koniec

P_0
P_1
P_2

Praca

A	B	C

Algorytm wykrywania zakleszczeń — przykład

Przydzielone

	A	B	C
P_0	2	1	0
P_1	1	2	1
P_2	1	1	1

Zamówienia

	A	B	C
P_0	1	2	1
P_1	1	1	1
P_2	0	1	0

Dostępne

A	B	C
0	1	0

Koniec

P_0
P_1
P_2

Praca

A	B	C
0	1	0

Algorytm wykrywania zakleszczeń — przykład

Przydzielone

	A	B	C
P_0	2	1	0
P_1	1	2	1
P_2	1	1	1

Zamówienia

	A	B	C
P_0	1	2	1
P_1	1	1	1
P_2	0	1	0

Dostępne

A	B	C
0	1	0

Koniec

P_0	false
P_1	
P_2	

Praca

A	B	C
0	1	0

Algorytm wykrywania zakleszczeń — przykład

Przydzielone

	A	B	C
P_0	2	1	0
P_1	1	2	1
P_2	1	1	1

Zamówienia

	A	B	C
P_0	1	2	1
P_1	1	1	1
P_2	0	1	0

Dostępne

A	B	C
0	1	0

Koniec

P_0	false
P_1	false
P_2	

Praca

A	B	C
0	1	0

Algorytm wykrywania zakleszczeń — przykład

Przydzielone

	A	B	C
P_0	2	1	0
P_1	1	2	1
P_2	1	1	1

Zamówienia

	A	B	C
P_0	1	2	1
P_1	1	1	1
P_2	0	1	0

Dostępne

A	B	C
0	1	0

Koniec

P_0	false
P_1	false
P_2	false

Praca

A	B	C
0	1	0

Algorytm wykrywania zakleszczeń — przykład

Przydzielone

	A	B	C
P_0	2	1	0
P_1	1	2	1
P_2	1	1	1

Zamówienia

	A	B	C
P_0	1	2	1
P_1	1	1	1
P_2	0	1	0

Dostępne

A	B	C
0	1	0

Koniec

P_0	false
P_1	false
P_2	false

Praca

A	B	C
0	1	0

Algorytm wykrywania zakleszczeń — przykład

Przydzielone

	A	B	C
P_0	2	1	0
P_1	1	2	1
P_2	1	1	1

Zamówienia

	A	B	C
P_0	1	2	1
P_1	1	1	1
P_2	0	1	0

Dostępne

A	B	C
0	1	0

Koniec

P_0	false
P_1	false
P_2	false

Praca

A	B	C
0	1	0

Algorytm wykrywania zakleszczeń — przykład

Przydzielone

	A	B	C
P_0	2	1	0
P_1	1	2	1
P_2	1	1	1

Zamówienia

	A	B	C
P_0	1	2	1
P_1	1	1	1
P_2	0	1	0

Dostępne

A	B	C
0	1	0

Koniec

P_0	false
P_1	false
P_2	true

Praca

A	B	C
1	2	1

Algorytm wykrywania zakleszczeń — przykład

Przydzielone

	A	B	C
P_0	2	1	0
P_1	1	2	1
P_2	1	1	1

Zamówienia

	A	B	C
P_0	1	2	1
P_1	1	1	1
P_2	0	1	0

Dostępne

A	B	C
0	1	0

Koniec

P_0	false
P_1	false
P_2	true

Praca

A	B	C
1	2	1

Algorytm wykrywania zakleszczeń — przykład

Przydzielone

	A	B	C
P_0	2	1	0
P_1	1	2	1
P_2	1	1	1

Zamówienia

	A	B	C
P_0	1	2	1
P_1	1	1	1
P_2	0	1	0

Dostępne

A	B	C
0	1	0

Koniec

P_0	false
P_1	false
P_2	true

Praca

A	B	C
1	2	1

Algorytm wykrywania zakleszczeń — przykład

Przydzielone

	A	B	C
P_0	2	1	0
P_1	1	2	1
P_2	1	1	1

Zamówienia

	A	B	C
P_0	1	2	1
P_1	1	1	1
P_2	0	1	0

Dostępne

A	B	C
0	1	0

Koniec

P_0	true
P_1	false
P_2	true

Praca

A	B	C
3	3	1

Algorytm wykrywania zakleszczeń — przykład

Przydzielone

	A	B	C
P_0	2	1	0
P_1	1	2	1
P_2	1	1	1

Zamówienia

	A	B	C
P_0	1	2	1
P_1	1	1	1
P_2	0	1	0

Dostępne

A	B	C
0	1	0

Koniec

P_0	true
P_1	false
P_2	true

Praca

A	B	C
3	3	1

Algorytm wykrywania zakleszczeń — przykład

Przydzielone

	A	B	C
P_0	2	1	0
P_1	1	2	1
P_2	1	1	1

Zamówienia

	A	B	C
P_0	1	2	1
P_1	1	1	1
P_2	0	1	0

Dostępne

A	B	C
0	1	0

Koniec

P_0	true
P_1	false
P_2	true

Praca

A	B	C
3	3	1

Algorytm wykrywania zakleszczeń — przykład

Przydzielone

	A	B	C
P_0	2	1	0
P_1	1	2	1
P_2	1	1	1

Zamówienia

	A	B	C
P_0	1	2	1
P_1	1	1	1
P_2	0	1	0

Dostępne

A	B	C
0	1	0

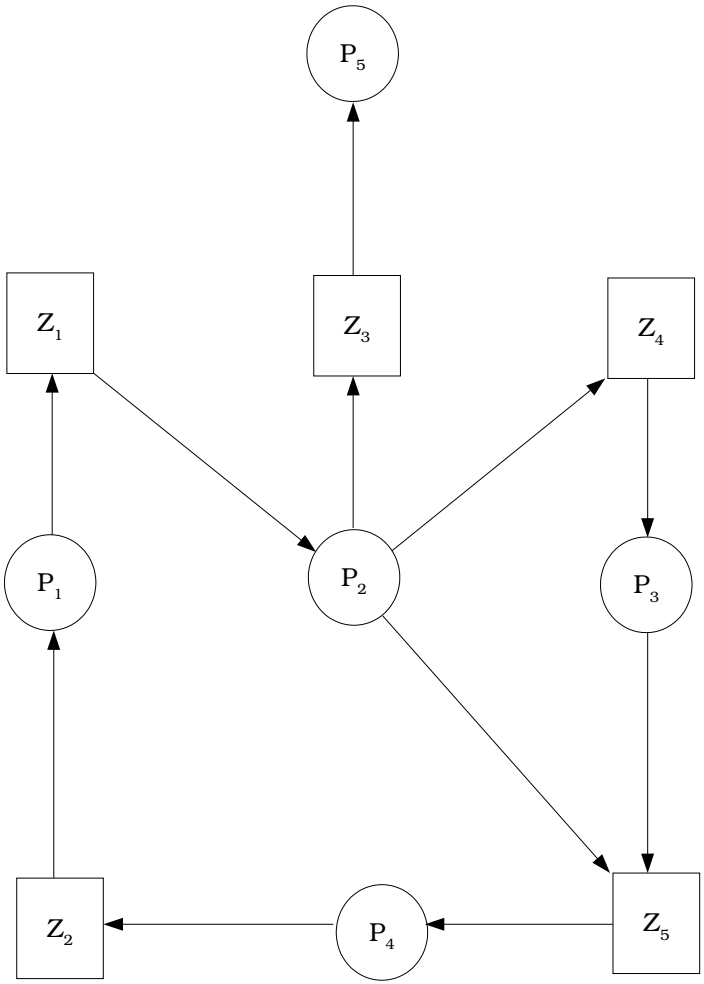
Koniec

P_0	true
P_1	true
P_2	true

Praca

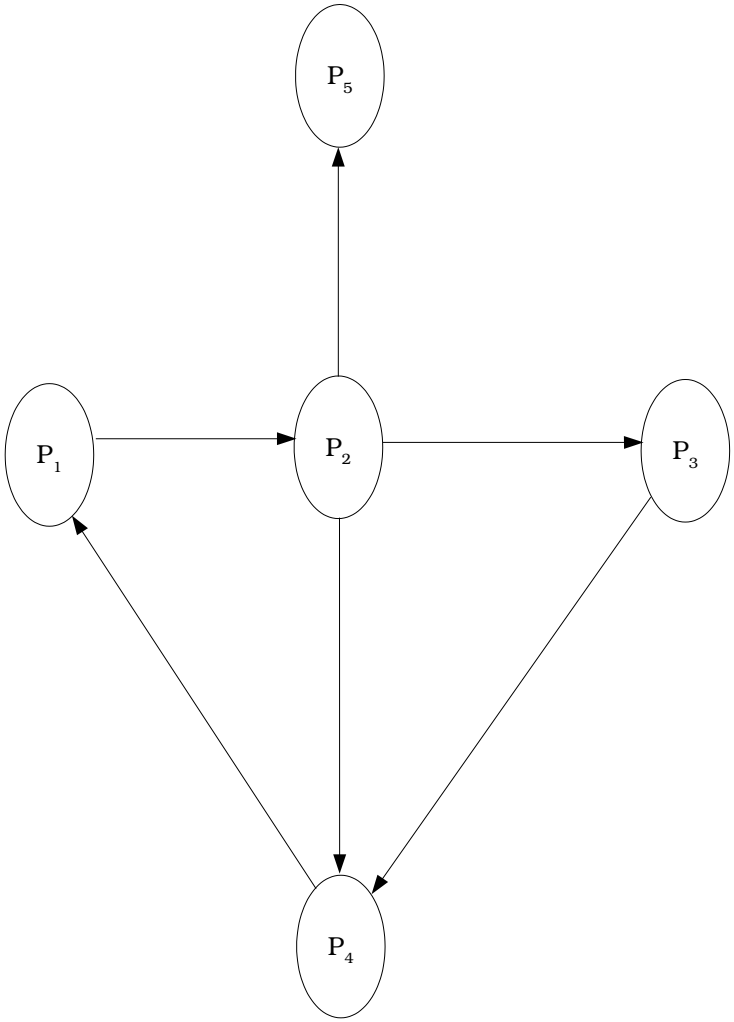
A	B	C
4	5	2

Algorytm wykrywania zakleszczeń dla zasobów reprezentowanych pojedynczo



Algorytm wykrywania zakleszczeń można uprościć dla systemów, w których występuje tylko jeden egzemplarz zasobu danego typu. Można go sprowadzić do algorytmu wykrywania cykli w *grafie oczekiwań*. Taki graf powstaje z przekształcenia grafu przydziału zasobów, poprzez usunięcie wierzchołków reprezentujących typy zasobów i połączenie powstałych w ten sposób wolnych krawędzi.

Algorytm wykrywania zakleszczeń dla zasobów reprezentowanych pojedynczo



Algorytm wykrywania zakleszczeń można uprościć dla systemów, w których występuje tylko jeden egzemplarz zasobu danego typu. Można go sprowadzić do algorytmu wykrywania cykli w *grafie oczekiwań*. Taki graf powstaje z przekształcenia grafu przydziału zasobów, poprzez usunięcie wierzchołków reprezentujących typy zasobów i połączenie powstałych w ten sposób wolnych krawędzi.

Korzystanie z algorytmu wykrywania zakleszczeń

Jeśli algorytm wykrywania zakleszczeń ma zostać zastosowany w rzeczywistym systemie, to należy odpowiedzieć na pytanie: „Jak często należy go wykonywać?" Przewrotna odpowiedź brzmi: „Częściej niż występują zakleszczenia." W praktyce może to oznaczać, że algorytm ten będzie wykonywany po każdym żądaniu przydziału zasobu. To z kolei może prowadzić do powstania dużych narzutów czasowych, związanych z jego wykonaniem. Inne podejście może bazować na mierze przepustowości systemu. Jeśli spadnie ona poniżej określonego progu, to należy algorytm wykrywania zakleszczeń wykonać. Przy tym rozwiązaniu może okazać się niemożliwe zidentyfikowanie sprawców zakleszczenia.

Wychodzenie z zakleszczenia

Po wykryciu zakleszczenia system operacyjny może powiadomić użytkownika i jemu pozostawić usunięcie tego problemu. Może również sam podjąć kroki zmierzające do wydostania się z zakleszczenia. Wszystkie strategie wychodzenia z zakleszczenia można podzielić na dwie kategorie:

1. zakańczanie procesów,
2. wywłaszczanie zasobów.

Kolejne plansze zawierają omówienie tych metod.

Wychodzenie z zakleszczenia

Po wykryciu zakleszczenia system operacyjny może powiadomić użytkownika i jemu pozostawić usunięcie tego problemu. Może również sam podjąć kroki zmierzające do wydostania się z zakleszczenia. Wszystkie strategie wychodzenia z zakleszczenia można podzielić na dwie kategorie:

1. zakańczanie procesów,
2. wywłaszczanie zasobów.

Kolejne plansze zawierają omówienie tych metod.

Wychodzenie z zakleszczenia

Po wykryciu zakleszczenia system operacyjny może powiadomić użytkownika i jemu pozostawić usunięcie tego problemu. Może również sam podjąć kroki zmierzające do wydostania się z zakleszczenia. Wszystkie strategie wychodzenia z zakleszczenia można podzielić na dwie kategorie:

1. zakańczanie procesów,
2. wywłaszczanie zasobów.

Kolejne plansze zawierają omówienie tych metod.

Kończenie procesów

Operację kończenia procesów zakleszczonych można przeprowadzić na dwa sposoby:

1. **Usunięcie wszystkich procesów zakleszczonych.** To rozwiązanie jest dosyć radykalne, bowiem tracimy wyniki pracy wszystkich procesów, które zostaną usunięte. Aby je odzyskać, należy ponownie uruchomić wszystkie te procesy.
2. **Usuwanie pojedynczych procesów, które uległy zakleszczeniu.** W tej metodzie musimy po usunięciu każdego procesu sprawdzić, czy zakleszczenie nadal istnieje. Należy również określić kryterium według którego będzie wybierany kolejny proces do zakończenia. Najczęściej przyjmuje się jedno z następujących:
   - Priorytet procesu.
   - Bieżący i pozostały czas pracy procesu.
   - Zasoby posiadane przez proces.
   - Zasoby wymagane przez proces do zakończenia pracy.
   - Ile innych procesów trzeba będzie zakończyć wraz z nim.
   - Typ procesu (interaktywny lub wsadowy).

Kończenie procesów

Operację kończenia procesów zakleszczonych można przeprowadzić na dwa sposoby:

1. **Usunięcie wszystkich procesów zakleszczonych.** To rozwiązanie jest dosyć radykalne, bowiem tracimy wyniki pracy wszystkich procesów, które zostaną usunięte. Aby je odzyskać, należy ponownie uruchomić wszystkie te procesy.
2. **Usuwanie pojedynczych procesów, które uległy zakleszczeniu.** W tej metodzie musimy po usunięciu każdego procesu sprawdzić, czy zakleszczenie nadal istnieje. Należy również określić kryterium według którego będzie wybierany kolejny proces do zakończenia. Najczęściej przyjmuje się jedno z następujących:
   - Priorytet procesu.
   - Bieżący i pozostały czas pracy procesu.
   - Zasoby posiadane przez proces.
   - Zasoby wymagane przez proces do zakończenia pracy.
   - Ile innych procesów trzeba będzie zakończyć wraz z nim.
   - Typ procesu (interaktywny lub wsadowy).

Kończenie procesów

Operację kończenia procesów zakleszczonych można przeprowadzić na dwa sposoby:

1. **Usunięcie wszystkich procesów zakleszczonych.** To rozwiązanie jest dosyć radykalne, bowiem tracimy wyniki pracy wszystkich procesów, które zostaną usunięte. Aby je odzyskać, należy ponownie uruchomić wszystkie te procesy.
2. **Usuwanie pojedynczych procesów, które uległy zakleszczeniu.** W tej metodzie musimy po usunięciu każdego procesu sprawdzić, czy zakleszczenie nadal istnieje. Należy również określić kryterium według którego będzie wybierany kolejny proces do zakończenia. Najczęściej przyjmuje się jedno z następujących:
   - Priorytet procesu.
   - Bieżący i pozostały czas pracy procesu.
   - Zasoby posiadane przez proces.
   - Zasoby wymagane przez proces do zakończenia pracy.
   - Ile innych procesów trzeba będzie zakończyć wraz z nim.
   - Typ procesu (interaktywny lub wsadowy).

Kończenie procesów

Operację kończenia procesów zakleszczonych można przeprowadzić na dwa sposoby:

1. **Usunięcie wszystkich procesów zakleszczonych.** To rozwiązanie jest dosyć radykalne, bowiem tracimy wyniki pracy wszystkich procesów, które zostaną usunięte. Aby je odzyskać, należy ponownie uruchomić wszystkie te procesy.
2. **Usuwanie pojedynczych procesów, które uległy zakleszczeniu.** W tej metodzie musimy po usunięciu każdego procesu sprawdzić, czy zakleszczenie nadal istnieje. Należy również określić kryterium według którego będzie wybierany kolejny proces do zakończenia. Najczęściej przyjmuje się jedno z następujących:
   - Priorytet procesu.
   - Bieżący i pozostały czas pracy procesu.
   - Zasoby posiadane przez proces.
   - Zasoby wymagane przez proces do zakończenia pracy.
   - Ile innych procesów trzeba będzie zakończyć wraz z nim.
   - Typ procesu (interaktywny lub wsadowy).

Kończenie procesów

Operację kończenia procesów zakleszczonych można przeprowadzić na dwa sposoby:

1. **Usunięcie wszystkich procesów zakleszczonych.** To rozwiązanie jest dosyć radykalne, bowiem tracimy wyniki pracy wszystkich procesów, które zostaną usunięte. Aby je odzyskać, należy ponownie uruchomić wszystkie te procesy.
2. **Usuwanie pojedynczych procesów, które uległy zakleszczeniu.** W tej metodzie musimy po usunięciu każdego procesu sprawdzić, czy zakleszczenie nadal istnieje. Należy również określić kryterium według którego będzie wybierany kolejny proces do zakończenia. Najczęściej przyjmuje się jedno z następujących:
   - Priorytet procesu.
   - Bieżący i pozostały czas pracy procesu.
   - Zasoby posiadane przez proces.
   - Zasoby wymagane przez proces do zakończenia pracy.
   - Ile innych procesów trzeba będzie zakończyć wraz z nim.
   - Typ procesu (interaktywny lub wsadowy).

Kończenie procesów

Operację kończenia procesów zakleszczonych można przeprowadzić na dwa sposoby:

1. **Usunięcie wszystkich procesów zakleszczonych.** To rozwiązanie jest dosyć radykalne, bowiem tracimy wyniki pracy wszystkich procesów, które zostaną usunięte. Aby je odzyskać, należy ponownie uruchomić wszystkie te procesy.
2. **Usuwanie pojedynczych procesów, które uległy zakleszczeniu.** W tej metodzie musimy po usunięciu każdego procesu sprawdzić, czy zakleszczenie nadal istnieje. Należy również określić kryterium według którego będzie wybierany kolejny proces do zakończenia. Najczęściej przyjmuje się jedno z następujących:
   - Priorytet procesu.
   - Bieżący i pozostały czas pracy procesu.
   - Zasoby posiadane przez proces.
   - Zasoby wymagane przez proces do zakończenia pracy.
   - Ile innych procesów trzeba będzie zakończyć wraz z nim.
   - Typ procesu (interaktywny lub wsadowy).

Kończenie procesów

Operację kończenia procesów zakleszczonych można przeprowadzić na dwa sposoby:

1. **Usunięcie wszystkich procesów zakleszczonych.** To rozwiązanie jest dosyć radykalne, bowiem tracimy wyniki pracy wszystkich procesów, które zostaną usunięte. Aby je odzyskać, należy ponownie uruchomić wszystkie te procesy.
2. **Usuwanie pojedynczych procesów, które uległy zakleszczeniu.** W tej metodzie musimy po usunięciu każdego procesu sprawdzić, czy zakleszczenie nadal istnieje. Należy również określić kryterium według którego będzie wybierany kolejny proces do zakończenia. Najczęściej przyjmuje się jedno z następujących:
   - Priorytet procesu.
   - Bieżący i pozostały czas pracy procesu.
   - Zasoby posiadane przez proces.
   - Zasoby wymagane przez proces do zakończenia pracy.
   - Ile innych procesów trzeba będzie zakończyć wraz z nim.
   - Typ procesu (interaktywny lub wsadowy).

Kończenie procesów

Operację kończenia procesów zakleszczonych można przeprowadzić na dwa sposoby:

1. **Usunięcie wszystkich procesów zakleszczonych.** To rozwiązanie jest dosyć radykalne, bowiem tracimy wyniki pracy wszystkich procesów, które zostaną usunięte. Aby je odzyskać, należy ponownie uruchomić wszystkie te procesy.
2. **Usuwanie pojedynczych procesów, które uległy zakleszczeniu.** W tej metodzie musimy po usunięciu każdego procesu sprawdzić, czy zakleszczenie nadal istnieje. Należy również określić kryterium według którego będzie wybierany kolejny proces do zakończenia. Najczęściej przyjmuje się jedno z następujących:
   - Priorytet procesu.
   - Bieżący i pozostały czas pracy procesu.
   - Zasoby posiadane przez proces.
   - Zasoby wymagane przez proces do zakończenia pracy.
   - Ile innych procesów trzeba będzie zakończyć wraz z nim.
   - Typ procesu (interaktywny lub wsadowy).

Kończenie procesów

Operację kończenia procesów zakleszczonych można przeprowadzić na dwa sposoby:

1. **Usunięcie wszystkich procesów zakleszczonych.** To rozwiązanie jest dosyć radykalne, bowiem tracimy wyniki pracy wszystkich procesów, które zostaną usunięte. Aby je odzyskać, należy ponownie uruchomić wszystkie te procesy.
2. **Usuwanie pojedynczych procesów, które uległy zakleszczeniu.** W tej metodzie musimy po usunięciu każdego procesu sprawdzić, czy zakleszczenie nadal istnieje. Należy również określić kryterium według którego będzie wybierany kolejny proces do zakończenia. Najczęściej przyjmuje się jedno z następujących:
   - Priorytet procesu.
   - Bieżący i pozostały czas pracy procesu.
   - Zasoby posiadane przez proces.
   - Zasoby wymagane przez proces do zakończenia pracy.
   - Ile innych procesów trzeba będzie zakończyć wraz z nim.
   - Typ procesu (interaktywny lub wsadowy).

Wywłaszczanie zasobów

Ta metoda polega na odbieraniu zasobów procesom uczestniczącym w zakleszczeniu i przydzielaniu ich innym, do czasu wyjścia z zakleszczenia. Algorytm wychodzenia z zakleszczenia oparty o wywłaszczanie musi podejmować decyzje dotyczące:

- **Wyboru ofiary.** Należy wybrać taki proces, aby odebranie mu zasobów prowadziło do jak najmniejszych kosztów.
- **Wycofywanie i wznawianie procesu.** Proces, któremu odebrano zasoby nie może kontynuować swojej pracy. Jego wykonanie musi więc zostać cofnięte do punktu, w którym można je będzie bezpiecznie wznowić.
- **Głodzenie procesu.** Należy zadbać o to, aby wybór ofiary nie prowadził do głodzenia żadnego z procesów, które mogą uczestniczyć w zakleszczeniu.

Wywłaszczanie zasobów

Ta metoda polega na odbieraniu zasobów procesom uczestniczącym w zakleszczeniu i przydzielaniu ich innym, do czasu wyjścia z zakleszczenia. Algorytm wychodzenia z zakleszczenia oparty o wywłaszczanie musi podejmować decyzje dotyczące:

- **Wyboru ofiary.** Należy wybrać taki proces, aby odebranie mu zasobów prowadziło do jak najmniejszych kosztów.
- **Wycofywanie i wznawianie procesu.** Proces, któremu odebrano zasoby nie może kontynuować swojej pracy. Jego wykonanie musi więc zostać cofnięte do punktu, w którym można je będzie bezpiecznie wznowić.
- **Głodzenie procesu.** Należy zadbać o to, aby wybór ofiary nie prowadził do głodzenia żadnego z procesów, które mogą uczestniczyć w zakleszczeniu.

Wywłaszczanie zasobów

Ta metoda polega na odbieraniu zasobów procesom uczestniczącym w zakleszczeniu i przydzielaniu ich innym, do czasu wyjścia z zakleszczenia. Algorytm wychodzenia z zakleszczenia oparty o wywłaszczanie musi podejmować decyzje dotyczące:

- **Wyboru ofiary.** Należy wybrać taki proces, aby odebranie mu zasobów prowadziło do jak najmniejszych kosztów.
- **Wycofywanie i wznawianie procesu.** Proces, któremu odebrano zasoby nie może kontynuować swojej pracy. Jego wykonanie musi więc zostać cofnięte do punktu, w którym można je będzie bezpiecznie wznowić.
- **Głodzenie procesu.** Należy zadbać o to, aby wybór ofiary nie prowadził do głodzenia żadnego z procesów, które mogą uczestniczyć w zakleszczeniu.

Wywłaszczanie zasobów

Ta metoda polega na odbieraniu zasobów procesom uczestniczącym w zakleszczeniu i przydzielaniu ich innym, do czasu wyjścia z zakleszczenia. Algorytm wychodzenia z zakleszczenia oparty o wywłaszczanie musi podejmować decyzje dotyczące:

- **Wyboru ofiary.** Należy wybrać taki proces, aby odebranie mu zasobów prowadziło do jak najmniejszych kosztów.
- **Wycofywanie i wznawianie procesu.** Proces, któremu odebrano zasoby nie może kontynuować swojej pracy. Jego wykonanie musi więc zostać cofnięte do punktu, w którym można je będzie bezpiecznie wznowić.
- **Głodzenie procesu.** Należy zadbać o to, aby wybór ofiary nie prowadził do głodzenia żadnego z procesów, które mogą uczestniczyć w zakleszczeniu.

Łączenie metod radzenia sobie z zakleszczeniami

Metody radzenia sobie z zakleszczeniami nie są metodami wzajemnie się wykluczającymi. Może się więc okazać korzystne zastosowanie w systemie wszystkich tych metod równocześnie. Łączenia działania takich algorytmów dokonuje się określając dla poszczególnych grup zasobów inne scenariusze rozwiązywania problemu zakleszczeń.

Algorytm strusia

Wielu programistów systemowych stosuje podejście do problemu zakleszczeń, które nazywa się *algorytmem strusia*. Polega on po prostu na ignorowaniu problemu. Wbrew pozorom taka strategia nie jest bezpodstawna. Zakleszczenia mogą pojawiać się w systemie tak rzadko, że wprowadzenie mechanizmów zapobiegania, unikania bądź wykrywania i wychodzenia z zakleszczeń staje się nieekonomiczne, z uwagi na ograniczenia nakładane na system przez te mechanizmy.

Pytania

?

Koniec

Dziękuję Państwu za uwagę!